METODOLOGÍA PARA EL DISEÑO DE SISTEMAS EMBEBIDOS BASADA EN MODELOS DE REDES DE PETRI

M. Alcaraz-Mejía, R. Campos-Rodríguez, L. García-Contreras, J. Urquieta-Jiménez, M. Aguirre-Villa

Departamento de Ciencias Tecnológicas,
Centro Universitario de la Ciénega, Universidad de Guadalajara
Av. Universidad 1115, Col. Lindavista, Ocotlán, Jalisco, 47820, México
Tel. +52(392)9259400, ext. 8369, correo electrónico: {mildreth.alcaraz, raul.campos}@cuci.udg.mx

RESUMEN

En este trabajo se presenta una plataforma embebida para el control de sistemas de eventos discretos. El sistema real de propósito específico se modela como una Red de Petri con señales de entrada y salida. Esto permite adaptar un esquema de control que puede ejercer acción sobre el sistema mediante comandos de entrada y retroalimentar el estado mediante señales de salida. Se proporciona una metodología para traducir el modelo en Red de Petri a una implementación de firmware embebido.

Palabras clave: Sistema Embebido, Control, Sistemas de Eventos Discretos, Redes de Petri.

I. INTRODUCCIÓN

Los sistemas de eventos discretos se encuentran prácticamente en todos los aspectos de la vida cotidiana, entre los que se consideran los sistemas embebidos. Estos sistemas poseen varias características a las que se debe prestar especial atención. Por ejemplo, un sistema embebido se programa comúnmente una sola vez, es decir, no permite realizarle actualizaciones después que éste se encuentra trabajando. En los raros casos en los que esto se permite, resulta muy complicado y costoso. Otra característica crítica de los sistemas embebidos es que normalmente se utilizan para aplicaciones dónde la confiabilidad es un factor primordial [1], [2].

Debido a las características que presenta un sistema embebido, las técnicas de programación de software convencionales no son aptas para su utilización en el desarrollo de los mismos, ya que los sistemas embebidos requieren de pruebas y análisis más exhaustivos. Por ello, uno de los principales retos a los que se enfrentan los ingenieros que diseñan sistemas embebidos es que no existen metodologías formales relacionadas con los aspectos del diseño y la implementación de hardware, o la verificación del software necesario para realizar una adecuada implementación de estos sistemas [2].

A pesar de su importancia, la literatura relacionada con metodologías formales para el diseño de sistemas embebidos es relativamente nueva. En [3], el autor adopta un enfoque práctico sobre la implementación de sistemas embebidos basados en el lenguaje de programación C y C++. Ofrece una amplia gama de consejos y trucos prácticos basados en la experiencia del autor. En [4], los autores hacen una revisión de las metodologías formales para el diseño, la implementación y validación de sistemas embebidos. Sin embargo no ofrecen aspectos técnicos ni prácticos de implementación, ni técnicas para la traducción de dichas metodologías a código funcional. En [5], los autores se enfocan a la programación de sistemas embebidos de manera práctica. Abordan varios aspectos del diseño de sistemas embebidos, como la selección del procesador, tamaño de buses y periféricos de comunicación. Consideran el diseño del firmware basado en diagramas de flujo y pseudocódigo. En [12], los autores presentan un formalismo basado en Redes de Petri y Lógicas Temporales, para la verificación de sistemas embebidos. Sin embargo, la metodología para

Figura 1. Sistema Generalizado

pasar del modelo a la implementación no es directa. En [13], el autor se enfoca al diseño de controladores en hardware y software desde un punto de vista eminentemente práctico. Incluso considera el uso de un sistema operativo embebido para la realización de controladores complejos y difíciles de implementar en un entorno operativo diferente.

En este trabajo se propone una metodología basada en modelos en Redes de Petri para la implementación de sistemas embebidos. Se abordan aspectos del formalismo de las Redes de Petri y se consideran aspectos técnicos de su traducción a código ejecutable.

El resto de este trabajo se organiza de la siguiente manera. En la Sección II. se presentan los aspectos generales de los sistemas embebidos. En la Sección III. se introducen las nociones básicas de las Redes de Petri. En la Sección IV. se presenta la metodología de programación propuesta en este trabajo y se muestra un ejemplo de aplicación. Finalmente, en la Sección V. se ofrecen las conclusiones de este trabajo.

II. SISTEMAS EMBEBIDOS

Un Sistema Embebido es un sistema de propósito específico, que realiza sólo una o pocas funciones, generalmente corre en tiempo real, y se encuentra incrustado como parte de otro componente o sistema (no stand-alone) [1].

Los Sistemas Embebidos utilizan la potencia de un microcontrolador, como un MCU (microcontroller unit) o un DSC (digital signal controller) en su implementación particular. Estos microcontroladores combinan una unidad de procesamiento (como el CPU de la PC) con circuitos adicionales llamados periféricos, más otros circuitos en el mismo chip para hacer un pequeño módulo de control que requiere solo unos pocos componentes externos para formar un sistema completo.

Debido a su propósito específico, estos sistemas utilizan pocas capacidades de hardware, lo que permite reducir su tamaño considerablemente y, por lo tanto, su costo y consumo de energía. Sin embargo, esta reducción de hardware exige a su vez consideraciones extras en el diseño del código de control para garantizar un alto rendimiento [2]. Este código, se conoce como firmware debido a que se almacena en un chip tipo ROM (Read Only Memory) o en una memoria Flash.

Como se mencionó al inicio de esta sección, otra característica importante de estos sistemas es que presentan restricciones de tiempo real e interactúan en ambientes críticos, por lo que se debe aumentar al máximo su confiabilidad [2].

Un sistema embebido puede tener alguno(s) de los siguientes periféricos: un pequeño teclado, una pequeña pantalla, leds, botones, displays de un dígito o pocos caracteres, sensores de temperatura, de luz, de movimiento, de humo, entre otros [5].

Los sistemas embebidos son utilizados en diversas áreas, como en electrónica, equipo médico, sistemas de transporte, telecomunicaciones, electrodomésticos, seguridad, automatización, entre otras. La idea general de un sistema se presenta en la Figura 1. Esta consiste de una caja negra con entradas y salidas, las cuales pueden ser analógicas o digitales. En el caso general de un sistema embebido, como se muestra en la Figura 2, la caja principal cuenta con varios componentes de hardware, como el procesador, la memoria, e interfaces de comunicación [3].

III. TÉCNICA DE MODELADO

Las Redes de Petri constituyen una herramienta ideal para el diseño y análisis de sistemas de eventos discretos, como en el caso de los sistemas embebidos. Este modelo permite aplicar análisis formal para la verificación de propiedades que aseguraran el correcto diseño del sistema, tanto de hardware como de software, antes de pasar a la etapa de implementación. Al aplicar este tipo de análisis se ahorra mucho tiempo de depuración del código, y por consecuencia, reduce el costo final del sistema.

Para obtener el modelo matemático que represente al sistema, en este trabajo se utilizan las Redes de Petri Interpretadas (RPI) para la representación de sistemas con entradas y salidas [6], [7]. Las RPI son una extensión a las Redes de Petri (RP) [8], cuyo formalismo se presenta a continuación.

Definición 1. Una Estructura de Red de Petri (ERP) es un grafo bipartito $G = (P, T, I, O)$ con dos clases de nodos, los lugares $P = \{p_1, \dots, p_m\}$ y las transiciones $T = \{t_1, \dots, t_n\}$, y dos funciones de relación de flujo, la función de entrada $I: P \times T \to Z^+$ y la función de salida $O: P \times T \to Z^+$.

La ERP se puede representar convenientemente como una matriz, lo cual simplifica la representación matemática, así como las demostraciones formales.

Definición 2. Las matrices de incidencia, pre $C^-_{m\times n}$ y post $C^+_{m\times n}$, de una ERP (P, T, I, O), se definen como $C^-_{m\times n} = c_{ij}$, donde $c_{ij} := I(p_i, t_j)$, y $C^+_{m\times n} = c_{ij}$ donde $c_{ij} := O(p_i, t_j)$ para $1 \le i \le m, 1 \le j \le n$, respectivamente. La matriz de incidencia $C_{m\times n}$ se define como $C_{m\times n} = C^+_{m\times n} - C^-_{m\times n}$. Los índices m y n representan los números de lugares y transiciones en la ERP, respectivamente.

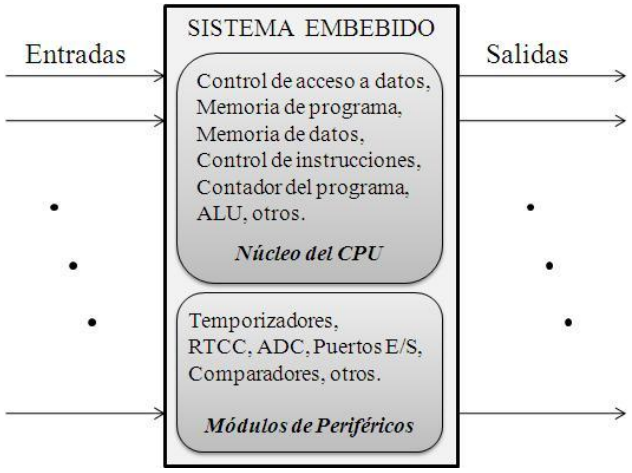

Figura 2. Sistema Embebido

El comportamiento dinámico de una RP se realiza a través de un juego de marcas. Una función de marcado $M: P \to Z^+$ representa el número de marcas en cada lugar.

Definición 3. Un Sistema de Red de Petri o simplemente una Red de Petri (PN) es un par (G, M_0) donde G es una ERP (P, T, I, O) y M_0 es una distribución inicial de marcas, conocida como Marcado Inicial, la cual representa las condiciones iniciales de la RP.

Las marcas en los lugares de la RP, permiten definir el comportamiento dinámico a través del disparo de las transiciones habilitadas. Informalmente, una transición puede ser disparada si existe un número suficiente de marcas en cada lugar de entrada en un momento dado.

Definición 4. Dada una transición $t_j \in T$ de una RP (Q, M_0), se dice que t_j está habilitada en el marcado M_k, denotado como $[M_k\rangle t_j$, si se cumple que $\forall p_i \in P, M_k(p_i) \ge I(p_i, t_j)$. El conjunto de todas las transiciones habilitadas en el marcado M_k se denota como $[M_k\rangle$.

El disparo de una transición habilitada en una RP permite alcanzar un nuevo marcado, el cuál puede ser calculado utilizando la ecuación de estados siguiente.

Definición 5. Una transición $t_j \in T$ habilitada en un marcado M_k, i.e., $[M_k\rangle t_j$, puede ser disparada alcanzando un nuevo marcado M_{k+1}, denotado como $M_k \xrightarrow{t_j} M_{k+1}$. El nuevo marcado M_{k+1} se calcula como:

$$M_{k+1} = M_k + C \cdot \vec{t_j} \qquad (1)$$

donde $\vec{t_j}$ es un vector de n-entradas, conocido como el vector de Parikh de la transición t_j, definido como $\vec{t_j}(i) = 0, i \ne j, \vec{t_j}(j) = 1$.

Definición 6. Una Red de Petri Interpretada (RPI) es un par (Q, M_0), donde

1) $Q = \{G, \Sigma, \lambda, \phi\}$, cuyos miembros se definen a continuación:

a) G es una ERP,
b) $\Sigma = \{\alpha_1, \ldots, \alpha_r\} \cup \{\varepsilon\}$ es un alfabeto de entrada, donde cada α_i es un símbolo de entrada y ε es el símbolo de nulidad,
c) $\lambda: T \to \Sigma$ es una función de etiquetado para transiciones,
d) $\varphi: R(G, M_0) \to (Z^+)^q$ es una función lineal de salidas que relaciona un marcado en $R(G, M_0)$ a un vector en $(Z^+)^q$, donde q es el número de salidas disponibles en la red;

2) M_0 es el marcado inicial de G.

En este trabajo, se considera que las funciones de salida y etiquetado φ y λ, son lineales, permitiendo su representación como matrices $\varphi_{q\times m}$ y $\lambda_{r\times n}$, respectivamente. El renglón i-ésimo $\varphi(i,:)$ de la matriz de salidas representa los lugares asociados a la señal de salida *i*-ésima, donde q y m son los números de señales de salida y el número de lugares en la red, respectivamente. Similarmente, el renglón *j*-ésimo $\lambda(j,:)$ de la matriz de entradas o etiquetado representa las transiciones asociadas a la señal de entrada *j*-ésima, donde r y n son el número de señales de entrada y el número de transiciones en la red, respectivamente.

Definición 7. Por definición de la función φ, un lugar $p_i \in P$ se dice ser *medible* siempre que $\varphi(:, i) \neq \vec{0}$, de otro modo es *no-medible.* Similarmente, por definición de la función λ, una transición $t_i \in T$ se dice ser *manipulable* si $\lambda(t_j) \neq \varepsilon$, de otro modo es *no-manipulable.*

Gráficamente, las transiciones no manipulables y los lugares no medibles se rellenan de gris.

Las señales de entrada permiten guiar "manualmente" o controlar el disparo de las transiciones. Esto se establece en la siguiente definición sobre el disparo de una transición en las RPI.

Definición 8. Sea $t_j \in T$ una transición en la RPI (Q, M_0) la cual está habilitada en el marcado M_k, i.e., $[M_k\rangle t_j$. El disparo de t_j depende de uno de los siguientes puntos:

- Si $\lambda(t_j) = \alpha_i \neq \varepsilon$ y el símbolo α_i se encuentra presente en el sistema, entonces t_j "*debe dispararse*";
- Si $\lambda(t_j) = \varepsilon$, entonces t_j "*puede dispararse*". En este caso, el disparo depende de la dinámica interna del sistema;
- En cualquier otro caso, t_j "*no debe dispararse*".

El incluir señales de entrada en una RPI permite selectivamente deshabilitar el disparo de las transiciones de la red, tal como se requiere en un esquema de control. La ecuación de estados para una RPI incluye adicionalmente a la ecuación de estados de una RP, una ecuación de salidas que considera la función de salida de la red.

Definición 9. Una transición habilitada $t_i \in T$ por el marcado M_k, cuyo símbolo $\lambda(t_i) = \alpha_i \neq \varepsilon$ está presente en el sistema, debe dispararse alcanzando un nuevo marcado M_{k+1} y produciendo un vector de salidas Y_{k+1}, tal como lo establece la siguiente ecuación:

$$M_{k+1} = M_k + C \cdot \vec{t_j}, \qquad y_{k+1} = \varphi \cdot M_{k+1} \tag{2}$$

donde y_{k+1} es el vector de salidas correspondiente al marcado M_{k+1} de la RPI.

Existen diversas metodologías de modelado de sistemas basadas en RP [7],[9]. La metodología presentada por Alcaraz et al. en [10], permite obtener una RPI, es decir, la RP junto con sus señales de entrada y salida.

En la Figura 3, se muestra una RPI (G, M_0), donde $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}\}$ y $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, t_{18}, t_{19}, t_{20}, t_{21}\}$. Las funciones I y O se representan con las matrices de incidencia Pre C^-_{mxn} y Post C^+_{mxn} siguientes, respectivamente:

$$C^-_{mxn} = \begin{bmatrix}
0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0\\
0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&1&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&0&0&1&0&0&0&0&1&0
\end{bmatrix},$$

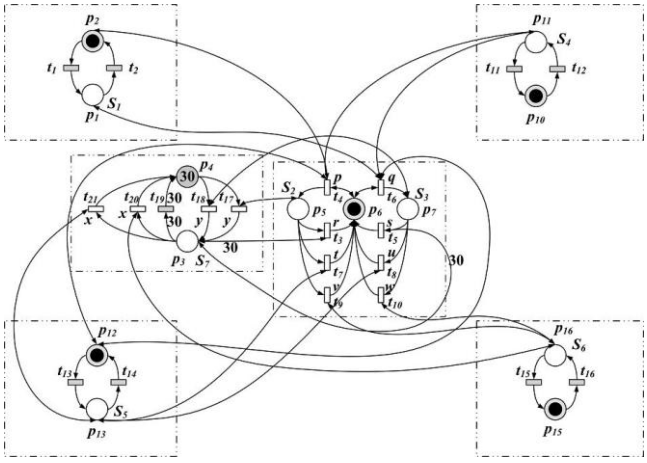

Figura 3. Red de Petri Interpretada

$$C^{+}_{mxn} = \begin{bmatrix} 1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1 \\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&1&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0&0&1 \\ 0&0&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&1&0&0&0&0&0 \end{bmatrix}$$

El marcado inicial de la RPI es $M_0 = 0\,1\,0\,30\,0\,1\,0\,1\,0\,0\,1\,1\,0T$

La función de salida es:

$$\varphi = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

El conjunto de símbolos de entrada es:

$\Sigma = \{p, q, r, s, u, x, y\}$

El etiquetado de las transiciones es el siguiente:

$\lambda(t_4) = p, \lambda(t_6) = q, \lambda(t_3) = \lambda(t_5) = r,$
$\lambda(t_7) = \lambda(t_8) = s, \lambda(t_9) = \lambda(t_{10}) = u, \lambda(t_{17}) = \lambda(t_{18}) = y, \lambda(t_{20}) = \lambda(t_{21}) = x, \lambda(t_1) = \lambda(t_2) = \lambda(t_{11}) = \lambda(t_{12}) = \lambda(t_{13}) = \lambda(t_{14}) = \lambda(t_{15}) = \lambda(t_{16}) = \lambda(t_{19}) = \varepsilon.$

IV. METODOLOGÍA DE PROGRAMACIÓN

En este trabajo se propone una metodología para traducir un modelo en RPI a su equivalente en programación embebida. Para ello debemos distinguir claramente las siguientes dos definiciones.

Definición 10. Una *Entrada* es aquella que se encuentra conectada al microcontrolador para que éste lea una señal o dato. Esta señal o dato es a su vez salida de otro sistema.

Definición 11. Una *Salida* es aquella que se encuentra conectada al microcontrolador para que éste escriba una señal o dato. Esta señal o dato es a su vez entrada de otro sistema.

Los pasos generales para convertir una RPI que representa un sistema de eventos discreto, a un programa de control embebido, son los siguientes:

1. Identificar las Entradas $C_1^e, C_2^e, \dots, C_n^e$.
2. Identificar las Salidas $C_1^s, C_2^s, \dots, C_m^s$.
3. Para cada Entrada $C_j^e, 1 \le j \le n$, asignar los bits de los Registros de los Puertos de Entrada $\{Rx1, Rx2, \dots, \dots\}$ a los que está conectado físicamente con el microcontrolador, y en los cuales se leerá la señal de entrada.
4. Para cada Salida $C_j^s, 1 \le j \le m$, asignar los bits de los Registros de los Puertos de Salida $\{Rx1, Rx2, \dots, \dots\}$ a los que está conectado físicamente con el microcontrolador, y en los cuales se escribirá la señal de salida.
5. Implementar la RPI con el formato mostrado en la Figura 4.

El procedimiento detallado para la implementación de la RPI se describe a continuación:

a. Definición de entradas y salidas. En esta parte es necesario definir con la etiqueta `#define` los bits del registro de control que se utiliza para cada entrada $C_j^e, 1 \le j \le m$. Opcionalmente, las entradas y salidas se pueden definir como constantes con `#define` y el `TRISx` para el registro x correspondiente.
b. Definir el número de lugares y el número de transiciones de la RPI con `#define`, cuyos valores son constantes durante la ejecución del programa.
c. Declarar las matrices de incidencia Pre y Post. Se recomienda declararlas para ser almacenadas en la memoria del código. Esto permite optimizar el uso de la memoria del

```
#include <picxxx.h>

// Definiciones de las entradas y salidas
#define Ce1         Rxa
#define Cs1         Rxb
     .
     .
// Definición de constantes de la RPI
#define NL          <Lugares>
#define NT          <Trans>

// Constantes en memoria de código
const unsigned char MatPre[NL][NT] = { { <I> }};
const unsigned char MatPost[NL][NT] = { { <O> }};
     .
     .
//Global Variable Declarations
unsigned char EstadoActual[NL]={ <Edo Inicial> };
unsigned char EstadoSiguiente[NL];
     .
     .
//Function Prototypes
void Inicializar();
void LeerEntradas();
unsigned char TransControlablesHabilitadas();
void EscribirSalidas();
void CalculoEstadoSiguiente();
unsigned char VerificacionErrores();
void CalcularEstadoActual();
void SeleccionarTransicion();
     .
     .
void main (void){
     Inicializar();
     while(1){
          CalcularEstadoActual();
          LeerEntradas();
          VerificacionErrores();
          if (TransControlablesHabilitadas()){
               SeleccionarTransicion();
               EscribirSalidas();
               CalcularEstadoSiguiente();
          }// fin if
     } //fin while
} //fin main
```

Figura 4. Código General de Implementación de una RPI

programa. Para ello se utiliza el comando `const` previo al tipo de dato definido para las matrices. En este caso se utiliza el `unsigned char`, que permite guardar números en el rango de [0..256]. Debe notarse que las matrices Pre y Post están formadas por números enteros positivos.

d. Declarar las variables globales. En este caso se sugiere declarar como variables globales el estado actual y el estado siguiente para la RPI, las cuales tienen tamaño NL (Número de Lugares).
e. Declarar los prototipos de las funciones a utilizar. Para este caso, se sugiere tener al menos las 8 funciones declaradas en el código que se muestra en la Figura 4.
f. Escribir la función principal (main) para la implementación de la RPI, como se muestra en la Figura 4.

El programa o función principal (`main`) se describe a continuación:

1. En la función `Inicializar()` se limpian todas las banderas, se limpian todas las entradas y salidas, se definen pines de entrada y salida (en caso de no incluirlas en la cabecera) se configuran temporizadores, entre otras.
2. Enseguida se inicia un ciclo mientras infinito (`while(1)`). Esto es, mientras el microcontrolador esté energizado, se deberá realizar lo que se encuentra dentro del ciclo.
3. La función `CalcularEstadoActual()` permite actualizar el estado actual de la RPI (`EstadoActual`), con el "estado siguiente" calculado previamente en la RPI (`EstadoSiguiente`). Incialmente, el estado actual es el marcado inicial.
4. La función `LeerEntradas()` realiza una lectura de los bits en los registros definidos como Entradas (bits = 1). Esto permite reconstruir el estado actual del Sistema Físico y compararlo contra el estado actual del modelo en RPI. En este caso, cada entrada leída es una salida en la RPI. Si el cambio se dio por una transición no-manipulable (evento externo), entonces se debe emplear algún mecanismo para reconstruir la trayectoria ejecutada por el sistema, en caso de que el esquema de control requiera retroalimentación de trayectorias. En

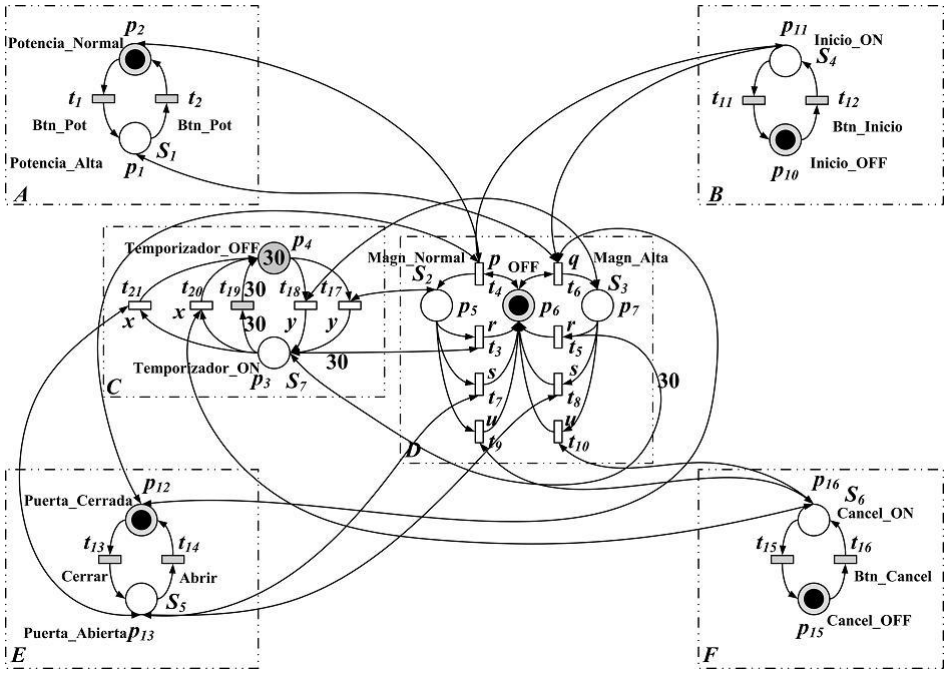

Figura 5. Modelo en RPI de la lógica de control de un horno de microondas simple

esta etapa del trabajo, no se aborda el tema reconstrucción de estados y trayectorias.

5. La función `VerificacionErrores()` es recomendable en la implementación de una RPI en sistemas embebidos. Esta realiza una comparación entre el estado actual del Sistema y el estado actual de la RPI, mediante la ecuación de salidas en (2). Estos estados deben ser equivalentes. De lo contrario, se deberá realizar un diagnóstico y, en su caso emplear un mecanismo de recuperación de errores. En esta etapa del trabajo, no se aborda el tema de diagnóstico y recuperación.
6. `TransControlablesHabilitadas()` permite verificar la existencia de transiciones manipulables habilitadas. Para esto, se utiliza la ecuación de habilitación en Definición 4 y la de manipulable en Definición 7.
7. La función `SeleccionarTransicion()` permite seleccionar una transición del conjunto de transiciones manipulables habilitadas para su disparo. En esta función se puede implementar un algoritmo de control, ya sea control por regulación [11] o control supervisor [8], el cual permitirá decidir acertadamente el siguiente evento a realizar.
8. La función `EscribirSalidas()` envía los comandos de control para realizar las actividades correspondientes a la transición seleccionada. El conjunto de comandos de control disponibles es Σ de la RPI, asignados a cada transición por λ.
9. `CalcularEstadoSiguiente()` utiliza la ecuación de estados de la RPI en (2) para calcular el estado siguiente de la RPI, después de la ejecución de los eventos correspondientes a la transición disparada.

Ejemplo de Aplicación

Para ilustrar el método de convertir la RPI en un programa de control embebido, utilizaremos como ejemplo un electrodoméstico de uso común en los hogares, el horno de microondas y un PIC de 16 bits. En la Figura 5 se encuentra el modelo en RPI de un horno de microondas. Cada recuadro A, B, C, D, E, F representa una variable de estado para los componentes Botón de Potencia, Botón de

Inicio, Temporizador, Magnetrón (generador de las microondas), Puerta del Horno, Botón de Cancelar, que intervienen en el sistema del horno de microondas, respectivamente. Los recuadros A, B, E y F representan el sistema de interfaz con el usuario. El recuadro D representa el dispositivo a controlar, es decir, el Magnetrón, y el recuadro C representa su temporizador. Cada sensor de salida asociado a cada componente del horno, son entradas necesarias para el control del magnetrón (generador de las microondas), mismas que están conectadas físicamente al PIC. Los únicos eventos controlables internamente son los asociados al magnetrón y el temporizador. Se debe observar que los eventos asociados a los dispositivos de la interfaz con el usuario, son controlados externamente por él. Es decir, son incontrolables para el PIC.

Con la finalidad de simplificar el ejemplo y hacerlo más didáctico, se consideran las siguientes hipótesis:

1. El horno (magnetrón) sólo funciona por tiempos de 30 segundos, y
2. Si se cancela o se abre la puerta, el horno deja de funcionar y el tiempo se reinicializa.

Para convertir la RPI de la Figura 5 a código embebido se utilizan los pasos descritos anteriormente. Estos son los siguientes:

1. Entradas al PIC: $\{S_1,\ S_2,\ S_3,\ S_4,\ S_5,\ S_6\}$
2. Salidas del PIC: $\{PrenderMagn\_N,\ PrenderMagn\_A,\ Reset,\ Tic\}$
3. $\{s_1 \rightarrow RB1,\ s_2 \rightarrow RB2,\ s_3 \rightarrow RB3,\ s_4 \rightarrow RB4,\ s_5 \rightarrow RB5,\ s_6 \rightarrow RB6\}$ con configuración del $TRISB = 0x003F$
4. $\{PrenderMagn_N \rightarrow RC1,\ PrenderMagn_A \rightarrow RC2,\ ApagarMagn_T \rightarrow RC3,\ ApagarMagn_P \rightarrow RC4,\ ApagarMagn_C \rightarrow RC5,\ Reset \rightarrow RC5,\ Tic \rightarrow RC6\}$ con configuración del $TRISC = 0x0000$
5. El resultado de la conversión al código embebido se muestra en la Figura 6.

A continuación se presenta una muestra de la ejecución del programa presentado.

1. Limpiar banderas (`INTCON = 0;`) ; Limpiar registros (`PORTB = 0; PORTC = 0;`); Definir pines de entrada y salida (`TRISB = 0x003F; TRISC = 0x0000;`); otras.
2. Calcula el estado actual. `EstadoActual` = $[0\ 1\ 0\ 30\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0]^T$
3. Leer las entradas `PORT_Read = PORTB`, inicialmente, los valores de `PORTB_Read = 0x0000`. Aquí se están leyendo todas las entradas al mismo tiempo.
4. Comparar la reconstrucción del estado actual del sistema real con el estado actual de la RPI. En este caso, sería $[0\ 1\ 0\ 30\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0]^T = [0\ 1\ 0\ 30\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0]^T$. Por lo tanto, no tenemos desviaciones.
5. No hay transiciones habilitadas.
6. En un ciclo posterior, suponiendo que un usuario pulsó el botón de inicio, entonces al leer las entradas, se tiene que `PORTB_Read = 0x0004`. La reconstrucción del estado es: $[0\ 1\ 0\ 30\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0]^T \neq [0\ 1\ 0\ 30\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0]^T$. Entonces, se disparó la transición no-manipulable t_{12}. Se debe actualizar el estado actual.
7. Ahora sí existen transiciones habilitadas manipulables.
8. Las transiciones habilitadas manipulables son: $\{t_4\}$, por lo tanto se selecciona ésta para su disparo.
9. Ahora, se debe realizar el evento correspondiente a la transición t_4: Prender el Magnetrón con Potencia Normal → `PORTC=0x0001`.
10. Ahora, el estado siguiente de la RPI es $[0\ 1\ 0\ 30\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0]^T$.

Este último marcado, permite el inicio del temporizador. Si no ocurre un evento externo, como que el usuario abra la puerta o pulse el botón de cancelar, entonces, cuando termine el

```
#include <pic24fj256gb106.h>

// Definiciones de las entradas y salidas
#define S1                RB1
#define S2                RB2
#define S3                RB3
#define S4                RB4
#define S5                RB5
#define S6                RB6
#define S7                RB7


#define PrenderMagn_N     RC1       //evento p
#define PrenderMagn_A     RC2       //evento q
#define ApagarMagn_T      RC3       //evento r
#define ApagarMagn_P      RC4       //evento s
#define ApagarMagn_C      RC5       //evento u
#define Reset             RC6       //evento x
#define Tic               RC7       //evento y

// Definición de constantes de la RPI
#define NL          13
#define NT          21

// Constantes en memoria de código
const unsigned char MatPre[NL][NT] =
     {{0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
      {1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
      {0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1}
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0}
      {0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0}
      {0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
      {0,0,0,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0}
      {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0}
      {0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0}
      {0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,1}
      {0,0,0,1,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0}
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0}
      {0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0}};

const unsigned char MatPost[NL][NT] =
     {{1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
      {0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
      {0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0}
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1}
      {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
      {0,0,1,0,1,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0}
      {0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
      {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0}
      {0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0}
      {0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,1}
      {0,0,0,1,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0}
      {0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0}
      {0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,1,0,0}};

//Global Variable Declarations
unsigned char EstadoActual[NL]={0,1,0,30,0,1,0,1,0,0,1,1,0}
unsigned char EstadoSiguiente[NL];
```

Figura 6. Parte del código de implementación de la RPI que representa un horno de microondas simple

temporizador, se tendrá que apagar el horno con el evento *r* de la transición t_3, es decir, `PORTC=0x0003`.

V. CONCLUSIONES

Se ha presentado una metodología para convertir una RPI a código de control para su implementación en un sistema embebido. Se introdujo detalladamente la noción de una RPI, se ejemplificó el diseño de modelos en esta técnica, y la forma en que estas pueden ser traducidas a código embebido. Se pudo observar que la traducción de RPI a código embebido es sencilla y directa. Se utilizó el PIC de 16 bits PIC24FJ256GB106 de Microchip® para la programación de la aplicación. Como parte del trabajo futuro se pretende agregar algún esquema de control y de recuperación de errores a la metodología.

VI. AGRADECIMIENTOS

Los autores agradecen a la Subsecretaría de Educación Superior e Investigación Científica por los apoyos de Fomento a la Generación y Aplicación Innovadora del Conocimiento y la Beca de Estudiante otorgados al proyecto "Metodologías para el Diseño de Sistemas Embebidos" mediante oficio PROMEP/103.5/08/2919.

VII. REFERENCIAS

[1]. Embedded Controller Hardware Design (Embedded Technology Series), by Ken Arnold. Publisher Newnes; 1st edition (January 15, 2001). ISBN-10: 1878707523.

[2]. Linux for Embedded and Real-Time Applications (Embedded Technology), by Doug Abbott. Publisher: Newnes; September 2002, ISBN-10: 0750675462.

[3]. Programming Embedded Systems in C and C++, by Michael Barr. O'Reilly Media, Inc.; 1999. ISBN 1565923545.

[4]. "Design of Embedded Systems: Formal Models, Validation and Synthesis", Stephen Edwards, Luciano Lavagno, Edward A. Lee, and Alberto Sangiovanni-Vincentelli; pp: 86-107 in *Readings in hardware/software co-design,* by Giovanni De Micheli, Rolf Ernst, Wayne Hendrix Wolf. Morgan Kaufmann; 1st edition, 2001. ISBN 1558607021.

[5]. Embedded Microprocessor Systems: Real World Design, Third Edition, by Stuart Ball,

Publisher: Newnes; 3rd edition, November 18, 2002, ISBN-10: 0750675349.

[6]. Free Choice Petri Nets (Cambridge Tracts in Theoretical Computer Science),by Jorg Desel and Javier Esparza, Publisher: Cambridge University Press; New Ed edition, September 8, 2005, ISBN-10: 0521019451.

[7]. Las redes de Petri: En la automática y la informática, por Manuel Silva Suárez, Editorial AC, 1985, ISBN-10: 8472880451, ISBN-16: 9788472880450.

[8]. Supervisory Control of Discrete Event Systems using Observers, R. Campos-Rodríguez, M. Alcaraz-Mejia, J. Mireles-García, Proceedings of the 15th Mediterranean Conference on Control & Automation. ISBN: 978-1-4244-1282-2, Julio, 2006. Atenas, Grecia.

[9]. Petri nets and manufacturing systems: An examples-driven tour, in Lectures on Concurrency and Petri Nets. Advances in Petri Nets, by L. Recalde and M. Silva and J. Ezpeleta and E. Teruel, Springer-Verlag, 2004.

[10]. Petri Net Based Fault Diagnosis of Discrete Event Systems, by Alcaraz-Mejia, M.; Lopez-Mellado, E.; Ramirez-Trevino, A.; Rivera-Rangel, I.; IEEE Conf. on Systems, Man and Cybernetics, 2003, Págs: 4730 – 4735, Vol. 5, ISBN: 1062-922X, October, 2003. Washington, DC, USA.

[11]. R. Campos-Rodríguez, A. Ramírez-Treviño, E. López-Mellado, "Regulation Control of Partially Observed Discrete Event Systems", en Proceeding of the 2004 IEEE International Conference on Systems, Man and Cybernetics, Vol. 2, Págs1837-1842.

[12]. Luis Alejandro Cortes, Luis Alej , Ro Corts, Petru Eles , Zebo Peng, "Verification of Embedded Systems using a Petri Net based Representation", in Proc. ISSS, 2000.

[13]. Embedded Controller Hardware Design, ARNOLD, K. (2001). Newnes Pub.; 1st edition 2001. ISBN 1878707523.

VIII. AUTORES

Dra. Mildreth Alcaraz Mejía es Ingeniero en Sistemas Computacionales por el Instituto Tecnológico de Colima, 2001. Obtuvo los títulos de Maestro y Doctor en Ciencias con especialidad en Ingeniería Eléctrica en el CINVESTAV Unidad Guadalajara, en el 2004 y 2007, respectivamente. Actualmente, se dedica a la aplicación y desarrollo de técnicas de control embebidas con una orientación hacia el control de los Microsistemas.

Dr. Raúl Campos Rodríguez es Ingeniero en Computación por el Centro Universitario de la Ciénega, UDG, 2000. Estudio Maestría y Doctorado en Ciencias en Ingeniería Eléctrica en el CINVESTAV Guadalajara, de 2002 a 2007. Trabaja activamente en el diseño de algoritmos para sistemas de eventos discretos y ha dedicado últimamente parte de su tiempo al diseño de Microsistemas, o MEMS, específicamente en mecanismos basados en fuerza electrostática y la expansión térmica.

Laura García Contreras es estudiante de la carrera de ingeniería en computación del 8° semestre en el Centro Universitario de la Ciénega, de la Universidad de Guadalajara. Trabaja en este proyecto como parte de su tesis de licenciatura.

Joel Urquieta Jiménez es estudiante de la carrera de ingeniería en computación del 8° semestre en el Centro Universitario de la Ciénega, de la Universidad de Guadalajara. Trabaja en este proyecto como parte de su tesis de licenciatura.

Martin Gerardo Aguirre Villa es estudiante de la carrera de ingeniería en computación del 8° semestre en el Centro Universitario de la Ciénega, de la Universidad de Guadalajara. Trabaja en este proyecto con una beca de PROMEP para estudiantes en proyectos de investigación.